XML Binding Specification and Validation

24 November 2015

In-line documentation in the transformation code should reference the rules that are being applied.

|  |  |  |
| --- | --- | --- |
|  | **Purpose** | Platform Independent Model to Platform Specific Model |
|  | **Description** | Flattening the Inheritance out of the Platform Independent Model |
|  | **File Location** | https://bitbucket.org/ddi-alliance/ddi-views/src/235a9c80cbf0956af4629c51a5e3a519a6537e3a/transform/XMLSchema/ |
|  | **File Name** |  |
|  | **Specification** | **Validation Criteria** | **Step** |
| 1 | Name conventions for classes, properties and relationships must follow the identified naming rules | Properties and relationships: lowerCamelCaseClasses : UpperCamelCase | Validate PIM |
| 2 | The following list of classes, properties and relationships must contain documentation stubs which are not empty | Property: Cardinality;DataType,DescriptionRelationship:TargetObject;Description;Source Cardinality;TargetCardinality;RelationshipType | Validate PIM |
| 3 | PIM will only contain those packages and functional views flagged for publication |  | Validate PIM |
| 4 | Abstract classes which function as extension bases have no properties or non-extension relationships | 1. If (PSM abstract class) properties = NULL
2. if (PSM abstract class) non-extension relationship=NULL
3. if (PSM abstract class) extension relationship=any(0,1)
 | TransformPIM to PSM |
| 5 | All properties and relationships are moved down the chain of inheritance to concrete classes | 1. if (PIM abstract class properties) = PSM concrete class properties down down the chain of inheritance
2. if (PIM abstract class relationships) = PSM concrete class properties down down the chain of inheritance
 | TransformPIM to PSM |
| 6 | When non-abstract classes extend non-abstract classes properties and relationships of the extension base are duplicated on the extending class | 1. if (PIM non-abstract class that extends another non-abstract class) = PSM properties are duplicated down the chain of inheritance
2. if (PIM non-abstract class that extends another non-abstract class) = PSM non-extension relationships are duplicated down the chain of inheritance
 | TransformPIM to PSM |

|  |  |  |
| --- | --- | --- |
|  | **Purpose** | Platform Specific Model transformation to XSD |
|  | **Description** | Production of a set of XSD schemas |
|  | **File Location** | https://bitbucket.org/ddi-alliance/ddi-views/src/235a9c80cbf0956af4629c51a5e3a519a6537e3a/transform/XMLSchema/ |
|  | **File Name** |  |
|  | **Specification** | **Validation Criteria** | **Step** |
| 1 | Normative schema for the library for all packages will be produced | Existence of schema | TransformPSM to XSD |
| 2 | Convenience (non-normative) schema for each functional view will be produced | Existence of schema | TransformPSM to XSD |
| 3 | Schemas must be compliant with the schema specification | ???? | PSM Validation |
| 4 | Schemas must be valid XML | Check using XML parser | PSM Validation |
| 5 | All schemas will be declared in the DDI namespace | Check the XML namespace attribute in each schema | PSM Validation |
| 6 | All schemas will import the XML namespace | Check the namespace imports for each schema | PSM Validation |
| 7 | The library schema will be named DDI\_Library.xsd | Check the schema is correctly named | PSM Validation |
| 8 | Each functional view schema will be named DDI\_[functional\_view\_name].xsd | Check the schema is correctly named | PSM Validation |
| 9 | The version of the schema[s] will be indicated in the body of the schema | Check the schema is correctly versioned | PSM Validation |
| 10 | Each enumeration class should be declared as a simple type | PSM enumerations should match simple type declarations in each schema | PSM Validation |
| 11 | Non-xsd primitive data types will be mapped to xsd primitive data types (UML) | Check simple types in schemas to ensure that no UML primitives are used | Transform PSM to XSD |
| 12 | The root element for each schema will be <DDI> with a type attribute containing the name of the functional view or the string library as appropriate | Check the root element of each schema and make sure the type attribute exists with the correct fixed value for each Functional View and the library | Transform PSM to XSD |
| 13 | For each class in the PSM, from the complex types package declare an xsd:ComplexType with child elements for each of the constructs within that complex type | Compare the complex datatypes in the PSM against declarations in the schema for the correct correspondence | Transform PSM to XSD |
| 14 | For each class in the PSM, from the complex datatypes package declare a global element using the xsd:ComplexType for that PSM complex data type | Compare the complex datatypes in the PSM against declarations in the schema for the correct correspondence | Transform PSM to XSD |
| 15 | For each class in other packages (non-complex datatype; these are classes which inherited from AnnotatedIdentifiable in the PIM) of the PSM generate an xsd:ComplexType where properties are expressed as child elements and non-inheritance relationships are expressed as references  | Compare non-complex datatype classes in the PSM against each schema for the correct correspondence | Transform PSM to XSD |
| 16 | Objects inside each Functional View are all declared in the non-normative convenience schema corresponding to that view | Compare the objects in each PSM Functional View for the correct correspondence in the non-normative corresponding schema | PSM Validation |
| 17 | Objects inside Library Packages are included in the normative library scheme | Compare the objects in the PSM library for the correct correspondence in the normative library schema | PSM Validation |
| 18 | Every property and relationship from the PSM for all classes should reflect the correct cardinalities in their xsd form | Check all the cardinalities for elements and attributes in the schemas against the corresponding construct in the PSM | PSM Validation |